Guide to Network Protocols

HTTP request

An HTTP request path is the part of the URL that comes after the domain name and indicates the specific resource being requested. For example, in the URL "http://www.example.com/products/widgets", the request path is "/products/widgets".

Here is an example of an HTTP GET request with a request path:

```
 GET /products/widgets HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
Accept-Language: en-US
Accept-Encoding: gzip, deflate
If-Modified-Since: Wed, 21 Oct 2015 07:28:00 GMT
```

This request is asking the server at www.example.com to retrieve the resource at "/products/widgets". The server can use this request path to determine which specific resource the client is requesting and return the appropriate data.

The request path can include additional information in the form of query parameters, which are key-value pairs that are appended to the end of the path and separated by a question mark (?). For example, the request path "/products/widgets?category=furniture&color=red" would include two query parameters: "category" with a value of "furniture", and "color" with a value of "red". The server can use these parameters to further refine the request and return more specific data.

HTTP response

Here is an example of an HTTP response to the GET request from the previous example:

```
 HTTP/1.1 200 OK
Date: Mon, 21 Jan 2022 15:47:32 GMT
Server: Apache/2.4.25 (Debian)
Last-Modified: Wed, 21 Oct 2015 07:28:00 GMT
ETag: "1d3-52407b3a72f59"
Accept-Ranges: bytes
Content-Length: 4653
Content-Type: text/html

<!DOCTYPE html>
<html>
  <head>
    <title>Widget Catalog</title>
  </head>
  <body>
    <h1>Widget Catalog</h1>
    <ul>
      <li>Red Widget</li>
      <li>Green Widget</li>
      <li>Blue Widget</li>
    </ul>
  </body>
</html>
```

This response consists of a status line, headers, and a message body.

The status line contains the HTTP version (HTTP/1.1), a status code (200), and a reason phrase (OK). The status code indicates the outcome of the request. A code of 200 means that the request was successful and the requested resource was returned.

The headers provide additional information about the response, such as the date and time it was sent, the server software that generated it, the size and type of the data being returned, and more.

The message body contains the actual data being returned in the response, in this case an HTML document containing a list of widgets.

MQTT: A Primer on the Lightweight IoT Protocol

MQTT (Message Queue Telemetry Transport) is a lightweight publish-subscribe messaging protocol that is used to send data between devices. It is designed to be simple and efficient, and is particularly well suited for IoT (Internet of Things) applications where low-powered devices need to send data over a network.

Here's how MQTT works:

1- A client device connects to an MQTT broker, a server that acts as a central message hub.

2- The client device can publish data to the broker by sending a message to a specific topic. A topic is a string that represents the subject of the message.

3- Other client devices that are subscribed to that topic will receive the message from the broker.

4- The client devices can also send messages to the broker requesting information or asking it to perform an action.

MQTT uses a publish-subscribe model, which means that clients can publish data to the broker without knowing who, if anyone, is subscribed to the data. This allows for a decoupled, asynchronous communication between devices.

MQTT has a number of features that make it well suited for IoT applications, such as low overhead, support for offline operation, and the ability to use different quality of service (QoS) levels to trade off between reliability and efficiency.

Componnents MQTT

The MQTT protocol consists of a number of components that work together to enable efficient and reliable communication between devices. Here are the main components of the MQTT protocol:

Clients

These are the devices that want to send or receive data using MQTT. Clients can be anything from sensors and actuators to smartphones and computers.

Broker

The broker is a server that acts as a central message hub. It is responsible for receiving data from clients, storing it, and forwarding it to other clients that are subscribed to the relevant topics.

Topics

Topics are used to categorize and route messages. Each message published by a client includes a topic, and clients can subscribe to specific topics to receive only the messages that they are interested in.

Quality of Service (QoS)

MQTT supports three levels of QoS, which determine how hard the broker and clients will try to ensure that a message is delivered. The levels are:

- QoS 0 (at most once): The broker will make a best effort to deliver the message, but does not guarantee that it will be

delivered.
- QoS 1 (at least once): The broker will guarantee that the message will be delivered at least once, but it may be delivered multiple times.
- QoS 2 (exactly once): The broker will guarantee that the message will be delivered exactly once. This is the most reliable but also the most resource-intensive level.
- Will Message: A will message is a message that is sent by the broker on behalf of a client if the client disconnects unexpectedly. The will message can be used to notify other clients that the client is no longer available.

These are the main components of the MQTT protocol. Together, they provide a simple and efficient way for devices to communicate with each other over a network.

Essay with MQTT

Client publish to a broker

Here is an example of a client publishing a message to an MQTT broker:

```
 Client: connect to broker at mqtt.example.com
Broker: connection accepted

Client: publish message "temperature: 72F" to topic "sensor/temperature"
Broker: store message and forward to subscribed clients
```

In this example, the client connects to the MQTT broker at mqtt.example.com and then publishes a message containing the current temperature to the topic "sensor/temperature". The broker receives the message and stores it, and then forwards it to any clients that are subscribed to the "sensor/temperature" topic.

Client subscribe to a topic

Here is an example of a client subscribing to a topic and receiving a message from the broker:

```
 Client: connect to broker at mqtt.example.com
Broker: connection accepted

Client: subscribe to topic "sensor/temperature"
Broker: send any stored messages for the "sensor/temperature" topic to the client

Broker: receive message "temperature: 72F" for topic "sensor/temperature" from a different client
Broker: forward message to subscribed clients, including the first client

Client: receive message "temperature: 72F"
```

In this example, the client connects to the broker and subscribes to the "sensor/temperature" topic. The broker sends any stored messages for that topic to the client, and then continues to forward new messages as they are received from other clients. In this case, the client receives a message containing the current temperature.

publish-subscribe pattern

The publish-subscribe pattern allows for a decoupled, asynchronous communication between devices. Clients do not need to know the identity or even the existence of other clients to communicate with them. This makes it easy to add new devices to the network or remove existing ones without disrupting the communication between the remaining devices.

Additionnal info on MQTT

Here are a few additional things to consider when using MQTT:

- MQTT supports three levels of quality of service (QoS) that determine how hard the broker and clients will try to ensure that a message is delivered. The levels are:
- QoS 0 (at most once): The broker will make a best effort to deliver the message, but does not guarantee that it will be delivered.
- QoS 1 (at least once): The broker will guarantee that the message will be delivered at least once, but it may be delivered multiple times.
- QoS 2 (exactly once): The broker will guarantee that the message will be delivered exactly once. This is the most reliable but also the most resource-intensive level.
- MQTT supports will messages, which are messages that are sent by the broker on behalf of a client if the client disconnects unexpectedly. The will message can be used to notify other clients that the client is no longer available.
- MQTT supports the use of retained messages, which are messages that are stored by the broker and sent to new subscribers when they subscribe to a topic. This can be useful for providing initial state information to new clients.
- MQTT supports the use of last will and testament (LWT) messages, which are will messages that are sent only if the client disconnects unexpectedly. This can be used to notify other clients of the client's unexpected disconnection.
- MQTT supports the use of authentication and encryption to secure the communication between clients and the broker.

MQTT VS HTTP

HTTP (Hypertext Transfer Protocol) and MQTT (Message Queue Telemetry Transport) are both protocols for sending and receiving data over a network, but they are designed for different purposes and have some key differences:

- Purpose: HTTP is a general-purpose protocol that is used to transfer data on the World Wide Web. It is used to request and retrieve web pages, images, and other resources from servers. MQTT is a lightweight messaging protocol that is designed specifically for IoT (Internet of Things) applications. It is used to send small data payloads between devices in a publish-subscribe model.
- Overhead: HTTP is a more verbose protocol than MQTT, which means that it requires more data to be transmitted to send a message. This can make it less efficient in situations where bandwidth or power are limited. MQTT is designed to be lightweight and efficient, with a smaller overhead than HTTP.
- Connection: HTTP uses a request-response model, in which a client sends a request to a server and the server sends a response. The connection is closed after the response is received. MQTT uses a publish-subscribe model, in which clients can publish data to a broker (server) and subscribe to specific topics to receive data. The connection remains open until the client disconnects.
- Reliability: HTTP provides a relatively reliable connection, but it does not guarantee that every request will be successfully received by the server or that every response will be successfully received by the client. MQTT provides three levels of quality of service (QoS) that allow clients to trade off between reliability and efficiency.
- Use cases: HTTP is a widely used protocol that is well suited for transferring data on the World Wide Web. MQTT is specifically designed for IoT applications and is well suited for sending small data payloads between low-powered devices.
- Client/serveur, comme HTTP.
- Publication et abonnement, comme MQTT.

So

It is not accurate to say that one protocol is "better" than the other, as it depends on the specific use case and requirements. Both HTTP and MQTT have their own strengths and are suitable for different types of applications.

HTTP (Hypertext Transfer Protocol) is a widely used protocol for transferring data on the World Wide Web. It is a general-purpose protocol that is suitable for a wide range of applications, including retrieving web pages, images, and other resources from servers. HTTP is relatively reliable and provides a request-response model, in which a client sends a request to a server and the server sends a response.

MQTT (Message Queue Telemetry Transport) is a lightweight messaging protocol that is specifically designed for IoT (Internet of Things) applications. It is used to send small data payloads between devices in a publish-subscribe model. MQTT is designed to be efficient and easy to implement on low-powered devices, and it provides three levels of quality of service (QoS) to allow clients to trade off between reliability and efficiency.

In general, HTTP is better suited for applications that require a more verbose and reliable protocol, such as transferring data on the World Wide Web. MQTT is better suited for applications that require a lightweight and efficient protocol, such as sending small data payloads between low-powered devices in an IoT network.

It is important to choose the right protocol for your specific use case, taking into account factors such as the type and amount of data being transferred, the reliability and overhead requirements, and the capabilities of the devices involved.

Further, HTTPs

HTTPS (Hypertext Transfer Protocol Secure) is a protocol that is based on HTTP (Hypertext Transfer Protocol). It is a secure version of HTTP that is used to transmit sensitive data, such as login credentials and financial information, over the internet.

Like HTTP, HTTPS is used to request and retrieve resources from servers. The main difference between the two protocols is that HTTPS uses encryption to secure the communication between the client and the server. This prevents third parties from intercepting and reading the data being transmitted.

To establish a secure connection using HTTPS, the client and the server must agree on a set of cryptographic keys to use for encryption. This process is known as the SSL/TLS (Secure Sockets Layer/Transport Layer Security) handshake. Once the keys are agreed upon, the client and server can communicate securely using HTTPS.

HTTPS is widely used on the World Wide Web to protect the privacy and security of users. It is especially important for websites that handle sensitive information, such as online banking, e-commerce, and social media.